Implementation and Demo

Advanced Software Engineering (COMS 4156)
Prof. Gail Kaiser

Team Warriors

Amy Jiang (rj2394)

Wendy Pan (wp2213)

Shreya Vaidyanathan (sv2525)

Andy Xu (lx2180)

Challenges in Implementation

Frontend

(Pair: Andy Xu (lx2180), and Wendy Pan (wp2213))

- **Understanding the React Model**: As opposed to MVC taught in class, React.js, the front-end framework we are using, has a very different model where components interact with each other through states and properties. We often had to encounter decisions including, for example, "should we put user login state in component A or in component A's parent and have component A be controlled by its properties", "should I separate out this "nested" JSX object and make it another stateless component?",etc.
- **Maintaining React Component States:** Separating the stateful parts and stateless parts is always a hard design decision. We often have to migrate states from one component to the other when we want to implement a new functionality/component. For example, initially, we only had the Login component, which maintained the user states (such as user credential and token), once we implement the Register component, user states were migrated to the home component, which is the common ancestor of Register and Login.
- **Async Handlers:** Handling asynchronous events turned out to be quite complicated and was the most time-consuming part for us, especially for update handlers that query the backend for user data. We tried out different XHttpRequest libraries but they all seem to be verbose and hard to manipulate in the context of React. After several trials, we ended up using d3.requests library as an alternative to the browser's built-in XMLHttpRequest library for its simplicity and support for lambda functions.
- **Loaders and Bundlers:** Although we set up the React framework and infrastructures at an early stage, we still encountered quite a few compiler issues with babel and CSS. As we add more libraries to React, they all need to be compiled and bundled by babel and webpack cohesively. We attempted to use material-ui as our UI framework but it did not get compiled and imported successfully with other React components. Since UI design is not a major concern in this class, we chose to fall back to react-bootstrap for simplicity and resolve the compiler issue.

Backend

(Pair: Ruoxin (Amy) Jiang (rj2394), and Shreya Vaidyanathan (sv2525))

- **Authorization and Authentication:** our application requires authentication to verify if a user is a registered vendor, and requires authorization to enforce correct policies for

different roles, i.e. a vendor can only manage his/her own postings, not others; a customer can only view the posts. We first attempted to implement the functionality using **oauth2.0** (verification with google account)**.** However, implementing oauth2.0 is very challenging and error-prone: it's not very clear how to incorporate the instructions in google's official documentation with our Flask app; most online tutorials/implementations have flaws -- some of them even embed the client secret in sample code. Finally, we decided to use ***JWT***, the implementation of which is much more straightforward and fits better with how our **RESTful** server behaves than oauth2.0.

- **Unit Testing:** another challenge in implementing the backend is to thoroughly test every component. The backend is quite complex: it currently has three models and four views and will expand as we add new features. To test the models, we use python unittest package, and to test the views, we use **flask-testing** which can simulate the frontend interactions (i.e http requests and responses). We have tried to cover many cases in unit testing and also use the testing code as documentation for the component under test.
- **REST API** : we have decided to implement the server as a REST service in order to improve the portability of the frontend to other platforms (mobile/app), increase the scalability and reliability of the project. Therefore, we have to make design choices and implement many things ourselves (without relying on original Flask framework) to follow the REST/HATEOAS principle (such as using JWT to remain stateless in the server).
- **Database Design**: we also encounter challenges in designing the database to efficiently perform the kinds of operations the backend will need to issue for each functionality. However we may need to rethink our design when more complex functionalities like querying nearby vendors by location are added.

Demo

Repo: https://github.com/ase-warriors/go-trucks/releases/tag/demo_v0.8

Current functionalities

- (Sign up) Vendors can create account. Each vendor will be assigned an unique vendor ID and an entry will be made in the database
- (Log in / Log out) Authorization and authentication
    - Vendors and customers need to follow different security policies (i.e. permitted resources/operations)
- Vendors can post from their account about their truck timings
    - This will be displayed publicly to any user accessing the system
- Users (and vendors) can view all postings (# to be fixed)

Functionalities to be implemented (for entire first iteration):

- Let the users to view vendor postings as points on a Google map.
- Support “searching nearby vendors by location” functionality in backend

Suggestions from Mentor/TA

- In the vendor’s dashboard, show only his/her own posts, not all postings; and place “browsing all listings” functionality at a different endpoint which is visible to all (users and vendors). This improvement will enforce a better separation between the roles (“user” and “vendor”).